Combining UML and SDL

Kurt Verschaeve

System and Software Engineering Lab, Vrije Universiteit Brussel, Pleinlaan 2, B 1050 Brussel, Belgium. kaversch@info.vub.ac.be

UML and SDL both have their own merits and form a strong alliance when combined in one methodology. In this paper we describe three scenarios for combining UML and SDL: Forward Engineering, Reverse Engineering and Round-Trip Engineering. The foundation that makes these scenarios possible is a profound translation between UML & SDL'96 concepts. We also explain how the translation is adapted to implement an incremental two-way translation between UML and SDL.

1. Introduction

SDL (Specification and Description Language) is a high-level specification and programming language. It is object-oriented, formal, and graphical. SDL is intended for the description of complex, event-driven, real-time, and communicating systems. Systems described in SDL consist of many processes running simultaneously which communicate with each other via signals. Each process is described by an extended finite state machine. SDL [EHS97] is standardized by ITU-T, as standard Z.100. SDL has a number of advantages compared to other high-level languages and to traditional low-level languages such as C, C++, or Java. SDL has a rich grammar that describes behavior and is unambiguous. Therefore, it is possible to build tools for the simulation of SDL systems and for the validation of formal characteristics, like deadlock avoidance. The SDL system can be translated into an executable application without manual coding, leading to shortened development time and increased quality. As a side effect, due to the readability of SDL diagrams, the SDL specification becomes the documentation in itself, ensuring simplified maintenance and post-development.

This paper describes the translation of an UML and SDL concepts and three scenario's how the mapping can be used to UML and SDL: forward engineering, reverse engineering and roundtrip engineering. Previous research on combining UML (or OMT) and SDL can be found in [VJW96], [Ver97] and [Ver99].

Our goal is to get the maximum profit of the advantages of both UML and SDL. UML and SDL share a number of qualities, like having a graphical notation, good readability and good tool support. They also incorporate object orientation and state machines, which make UML and SDL suitable to work together. But each of them also has enough advantages to make it worthwhile to use them both in one methodology. More specifically, UML provides generic concepts, has multiple views on the same information and poses little constraints during modeling with more flexibility. Moreover, there is a smooth transition from use cases, conceptual model and sequence diagrams to class diagrams and state charts. SDL on the other hand, provides specialized concepts, has a formal definition and semantics, is simulatable and executable and provides both a graphical and a textual syntax.

Figure 1 shows the common and unique diagrams and concepts available in UML and SDL. We come to the same conclusion that UML and SDL is a good alliance. They share the specification for the static structure, behavior and scenarios. Unique for UML is the use cases and the collaboration diagrams. In SDL the type specification and the transitions can be implemented in full detail. Note that UML Sequence Diagrams and MSC's both are used to specify scenarios, but are not dealt with in our round-trip engineering solution.

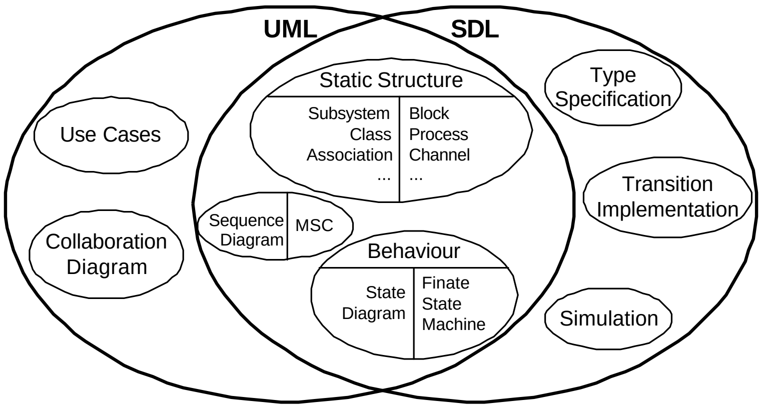


Figure 1. Comparison of Features of UML and SDL

The cross-section of Figure 1 shows the part of UML and SDL for which we have a translation. Based on this translation we can build support for three different scenarios:

1. Forward Engineering: This scenario is followed for new projects. The requirements analysis and system design is done in UML. The system design model is then translated to SDL, where the development continues with the round-trip scenario.
2. Reverse Engineering: This scenario is followed in the case that there is already an SDL specification available. The specification is translated to UML, either for documentation purposes or for reengineer purposes.
3. Round-trip Engineering: After either scenario 1 or 2, there is UML and SDL available for the same system. From then on the two models are kept synchronous by forwarding the changes made on the other side.

2. UML to SDL Translation

The mapping and translation between UML and SDL concepts is an essential part of our research. The mapping definition determines how the information in the UML model relates to information in the SDL system. Most mappings are syntax oriented, e.g. a UML class maps on an SDL process, but some mappings are more semantically, e.g. communication between classes maps on communication between processes. The translation of complete models or changes in the model is based on this mapping. We illustrate the translation of a full UML model with an example. We show the UML model of a Toffee Vendor and the resulting SDL specification after translation.

2.1 Static Structure

The basic building blocks of a model in UML are packages, subsystems and classes, where the classes represent the active components. In SDL the basic building blocks of a system are packages, blocks, block types, processes and process types. Another aspect of the static structure is the relationship between classes. Basically, associations map on communication in SDL and aggregation maps on nested structure. Figure 2 shows the improved class diagram of the toffee vendor example.

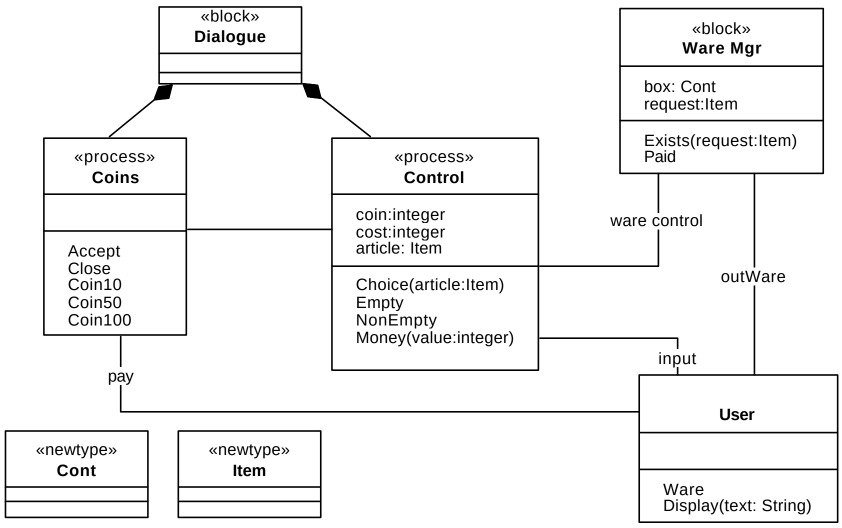
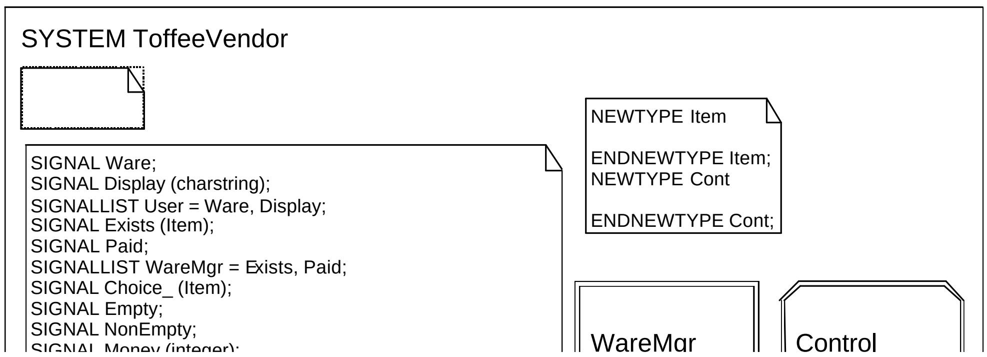


Figure 2. Static Structure of the Toffee Vendor

Figure 3 and Figure 4 shows the content of the generated toffee vendor system. Figure 3 shows the signal and type declarations. Two empty newtype declarations are created for the «newtype» classes *Item* and *Cont*. Figure 4 shows the block interaction at the system level. For every association between two classes, a communication path is generated between the corresponding processes or process instances. Channels are generated to reroute the communication path via the first common visible block. In our example, the channel ware_control is the result of rerouting the association between the classes Control and Ware_Mgr. The three other channels going to the environment are the result of the associations to the User class.



```
SIGNAL Ware;
SIGNAL Display (charstring);
SIGNALLIST User = Ware, Display;
SIGNAL Exists (Item);
SIGNAL Paid;
SIGNALLIST WareMgr = Exists, Paid;
SIGNAL Choice_ (Item);
SIGNAL Empty;
SIGNAL NonEmpty;
SIGNAL Money (integer);
```

```
NEWTYPE Item

ENDNEWTYPE Item;
NEWTYPE Cont

ENDNEWTYPE Cont;
```

Figure 3. Signal and Type Declarations in the Generated System

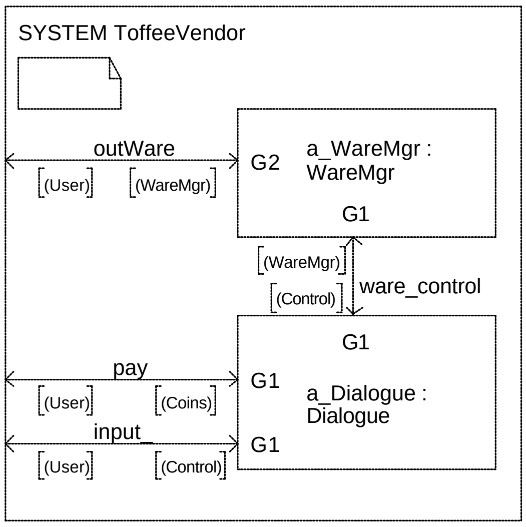


Figure 4. Block and Processing Interaction

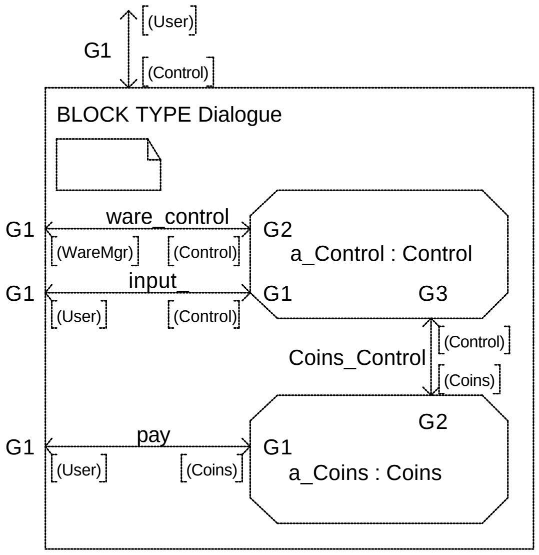


Figure 5. Process Interaction in Dialogue Block Type

2.2 State Diagrams

The translation of UML state diagrams to SDL state diagrams is rather straight forward, except for nested state diagrams and entry and exit actions. The UML state diagrams include the notion of nested hierarchical states, while SDL does not. This is solved by flattening the nested states and at the same time moves the entry and exit actions to the appropriate transitions. Figure 6 shows an example of a nested UML state diagram and its translation in SDL.

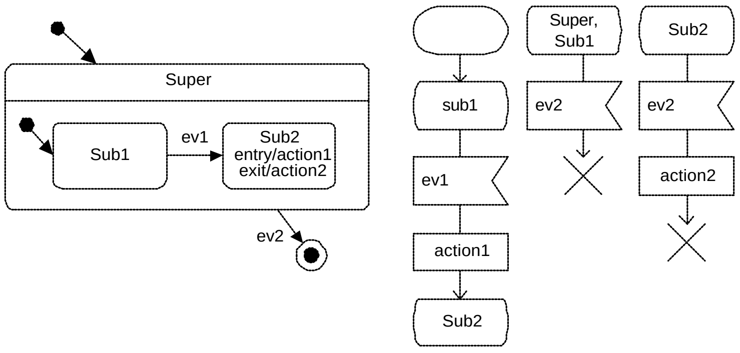


Figure 6: Mapping nested UML and SDL State Diagrams

3. Roundtrip Engineering

A one-shot translation of a full UML model to an SDL specification is only the first step. The next step is the possibility to develop concurrently in UML and SDL whereby both models are automatically kept synchronized. At each synchronization point, changes made on a higher level of abstraction are merged with the changes on the more concrete level or the other way around. The necessity for automated roundtrip engineering between UML and SDL is apparent. Without this support, maintenance will probably be done on the SDL level only because no one likes to make the same change twice. By not updating the UML model, it might be unclear what impact a change has on other parts of the system. Moreover, the UML model gets outdated and it becomes difficult to maintain the system.

Because is the mapping is not a strict one-on-one mapping, traditional roundtrip engineering solutions cannot be used. We propose to use a set of translation rules that define how changes in UML model are translated into changes in the SDL specification and the other way-around. Some examples of possible changes are: new class, rename operation, delete association, etc. These changes are automatically detected, translated to SDL (or UML) and applied locally on the specification with maximal preservation of detailed design changes in SDL. Hierarchical links between UML and SDL syntactic elements provide the context in the SDL system where to apply changes. The one-shot translation of a UML model to SDL is then seen as a comparison with an empty model, i.e. all entities in the model are “new”. This method does not work in the SDL to UML direction.

4. Conclusions

In this paper we argue that UML and SDL form a good alliance. Not only can UML contribute to the development of SDL based systems, SDL is also a good candidate to implement event-driven systems that are modeled in UML. Both the UML class diagrams and the state diagrams are mapped onto SDL concepts, which strengthen the cooperation.

The mapping definition forms the basis for a translation for UML to SDL and the other way round and for tool support for synchronizing a UML model and an SDL specification. This in turn makes it possible to combining UML and SDL is three scenarios: forward engineering, reverse engineering and round-trip engineering. To allow roundtrip engineering, we propose to translate changes instead of the full model as one piece. A large set of translation rules is necessary to translate any kind of change, e.g. class rename. The current state of our research is that we have over 180 rules to translate UML changes to SDL and about 70 rules to translate changes in SDL back to UML.

In the future UML and SDL will continue to grow toward each other. SDL 2000, not covered in this paper, really targets the integration with UML. For example, the SDL specification can partially be presented with UML syntax. Also some annoying restriction are be removed like the fact that block and process cannot be next to each and the difference between channel and signal route disappeared.

5. References

[EHS97] J. Ellsberger, D. Hogrefe, A. Sarma. *SDL, Formal Object-Oriented Language for Communicating Systems*. Prentice Hall, London, 1997.

[Ver97] Kurt Verschaeve. *Automated Iteration between OMT* and SDL*. 8th SDL Forum, Paris, 1997.

[Ver99] Three Scenarios for Combining UML and SDL'96, K. Verschaeve, A. Ek, Eighth SDL Forum, Montréal, Canada, 1999.

[VJW96] K. Verschaeve, V. Jonckers, B. Wydaeghe, L. Cuypers. *Translating OMT* to SDL, Coupling Object-Oriented Analysis with Formal Description Techniques*. Method Engineering 96 proceedings, p.126-141. IFIP, Atlanta, USA, 1996.